1st Report - Dynamic Function Exchange Basics

Dynamic Function eXchange (DFX)

DFX is the current name given by Vivado for Partial Reconfiguration (PR). This technique allows the reconfiguration of a defined module of the design at run time, enabling the system to continue its logic while a specific part of the hardware is changed.

As a result, the bitstream generation manages to create a top bit file and n partial bitstream files that share the top file. Due to that, it is possible to change the configuration isolating the inputs and outputs of the specific logic and connecting afterward because every configuration must share the same inputs and output.

Technical words for partial reconfiguration

Reconfigurable Partition (RP)	Instance responsible to allocate the multiple reconfigurations.
Reconfigurable Module (RM)	Each of the multiple reconfigurations that are allocated on the Reconfigurable Partition.
Static Logic / Top Level Logic	Everything besides the Reconfigurable Partitions.
Static Design	Everything that is not reconfigurable.
Partition Definition (PD)	Set of Reconfigurable Modules that will be allocated at a Reconfigurable Partition.

How to use it (1st basic tutorial)

In this tutorial, we will understand the basics and how Vivado IDE manages the creation and arranges partial reconfiguration.

1- Open Vivado and create a new project.

- Select your desired folder to allocate the project, make sure that the “Create project subdirectory” is checked. Click “Next”.
- Select RTL project and check on do not specify at this time. Click “Next”.
- Go to the board area and search for “Nexys4 DDR”, select the board. Then click “Next” and “Finish”.

In case you want to use the source files go directly to number 4.1, in case you want to generate them just continue.

Univ.Ass. Nahla El-Araby, PhD
Nilson Neves Filho, BSc

2 - Creating the top file.

- First click on the "+" button on the Sources panel, select "Add or create design sources". Then click next and again click on the "Create File" button. Select the desired language and file name. In this case, the file will be written on Verilog and the file name will be "top.v". Then Ok>Finish.

- Then a "Define Module" screen will appear, just click Ok>Yes.

3 - Defining Inputs and outputs

- In this case, we will create two Reconfigurable partitions, one called "shift" and one called "count". So we will set up first the top file inputs and outputs.

Inputs	Function	Outputs	Function
clk	responsible to provide the 100 Mhz signal from the board to each logic	[3:0] shift_out	leds that will receive the information from the 4 bits logic shift
reset	not reset signal from the board button	[3:0] count_out	leds that will receive the information from the 4 bits logic count

Then the top module I/Os will be:

```
module top(
   input clk,
   input reset,
   output [3:0] shift_out,
   output [3:0] count_out
   );
endmodule
```

- Now we will set the wires that will connect the inputs and outputs to the reconfigurable partition:

```
module top(
   input clk,
   input reset,
   output [3:0] shift_out,
   output [3:0] count_out
   );
```

```
    wire clk;
    wire reset;
    wire [3:0] shift_out;
    wire [3:0] count_out;

endmodule
```

4 - Creating the first logic.

- Click again on the “+” button on the sources area, then “Add or create design sources” > “Next” > “Create File”. First, we will create the shift_right file, so select the Verilog language and name the file as “shift_right” then click Ok>Finish>Ok>Yes.

- First, we will change the module name from shift_right to shift (due to similarity between the logics, the name and I/Os must be the same). Then we will define the I/Os for the logic:

```
module shift(
    input clk,
    input rst,
    output shift_out
    );
endmodule
```

- Now adding the logic.

```
module shift(
    input clk,
    input rst,
    output shift_out
    );

   reg [24:0] count; //new clock register responsible for making a low frequency clock
   reg [3:0]  shift_out; //register responsible to get the data and send it to the output
   reg [3:0]  shift_en = 4'b0101;  // Special init register to test REST_AFTER_RECONFIG

   //Counter to reduce speed of output
   always @(posedge clk)
      if (rst) begin
         count <= 0;
      end
      else begin
         count <= count + 1;
      end

    always @(posedge clk)
    if (rst)
      shift_out <= 4'b0000;
    else begin
      if (count == 25'b1111111111111111111111111 && shift_en == 4'b0101) begin //if the counter
achieve its maximum and the system were properly reconfigured
        case(shift_out)
        4'b0000 : shift_out=4'b1000;
        4'b1000 : shift_out=4'b0100;
        4'b0100 : shift_out=4'b0010;
        4'b0010 : shift_out=4'b0001;
        4'b0001 : shift_out=4'b1000;//led shifting logic
        endcase
      end
    end

endmodule
```

- Then we have to instantiate this module on the top.v:

```
module top(
   input sys_clk,
   input reset,
   output [3:0] shift_out,
   output [3:0] count_out
   );

   wire sys_clk;
   wire reset;
   wire [3:0] shift_out;
   wire [3:0] count_out;

   shift shift(
      .clk(sys_clk),
      .rst(~reset),
      .shift_out(shift_out)
   );

endmodule
```

- After defining both logic and the top file the result can be found in Annex 1.
- **4.1 Adding the source files (in case you did not want to creat the files):**
    - Click again on the “+” button on the sources area, then “Add or create design sources” > “Next” > “Add File”. Select “top.v”, “shift_right.v” and “count_up.v”, then click Ok>Finish>Ok>Yes.

5 - Creating the Partial Reconfiguration Files:

- First, click on Tools and Enable Dynamic Function eXchange> Convert. Then right-click on the shift_right file and select Create Partition Definition:
  Set Partition Definition Name: shift
  Reconfigurable Module Name: shift_right

- Do the same thing for count_up but set
  Partition Definition Name: count
  Reconfigurable Module Name: count_up
- In order to add the other configurations you have to click Tools>Dynamic Function eXchange Wizard>Next

- Click on "+"
  Partition Definition Name: shift
  Reconfiguration Module Name: shift_left

- Select + then add the "shift_left" file

- Do the same steps for count_down but define
  Partition Definition Name: count
  Reconfiguration Module Name: count_down

- Then click next and automatically create configurations. At this point, you can see that it is also possible to select "greybox" as a configuration. Feel free to create more configurations with this. Grey boxes are the same as black boxes, they attach the Reconfigurable Partition I/Os to the ground.

- Edit Configuration Runs, this part lets you adjust the configurations according to your specification for the first implementation and its children. Click in automatically create. Next>Finish.

6 - Contraints File:

- Now, we will add the project I/Os pins specification. In the case of Nexys 4 DDR the configuration of each pin can be found at Diligent's Git.
- Click again on the "+" button on the sources area, then "Add or create constraints" > "Next" > "Add File". Select "top_IO".

7 - Synthesis

- Run synthesis. In the current case, the constraints file already defines the Pblocks, which are the areas at the chip where there will be the Reconfiguration Partition.

- If you need to define the Pblocks, do the following steps:

- After Synthesis, open the Synthesized Project. Right-click at one reconfigurable partition. Pblock>Create Pblock. After that draw the Pblock and change its configurations RESET_AFTER_RECONFIG checked and ROUNTING ON. Do the same for each Reconfigurable Partition.

- Then Click on Report, select only Dynamic Function eXchange. This will check if the way that the Pblocks were arranged is valid for Partial Reconfiguration.

8 - Run Implementation

9 - Create Bitstreams

10 - Hardware Configuration

- With the Hardware Configuration Wizard, configure the board with the full bitstream with the impl_1. This is the first implementation file containing the first set of configurations. After that, you can configure any of the partial reconfiguration files that will change only one configuration at a time.